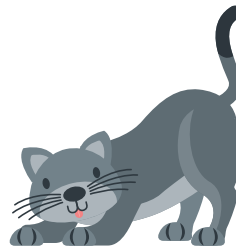
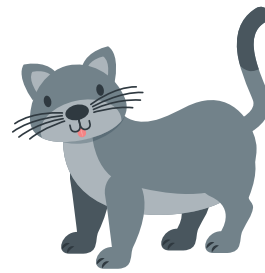
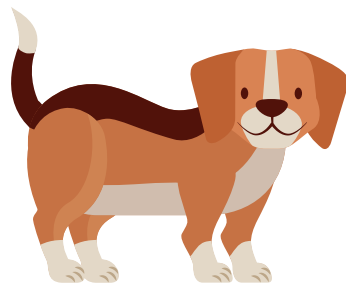


MIAUHEAVEN REFACTORING



*María Jesús Cadenas Sánchez
Sara Donaire Guillen
Alfonso González Fernández
Francisco Javier González Lugo
Rafael Salas Castizo
Javier Solís García*

REFACTORING

PetController

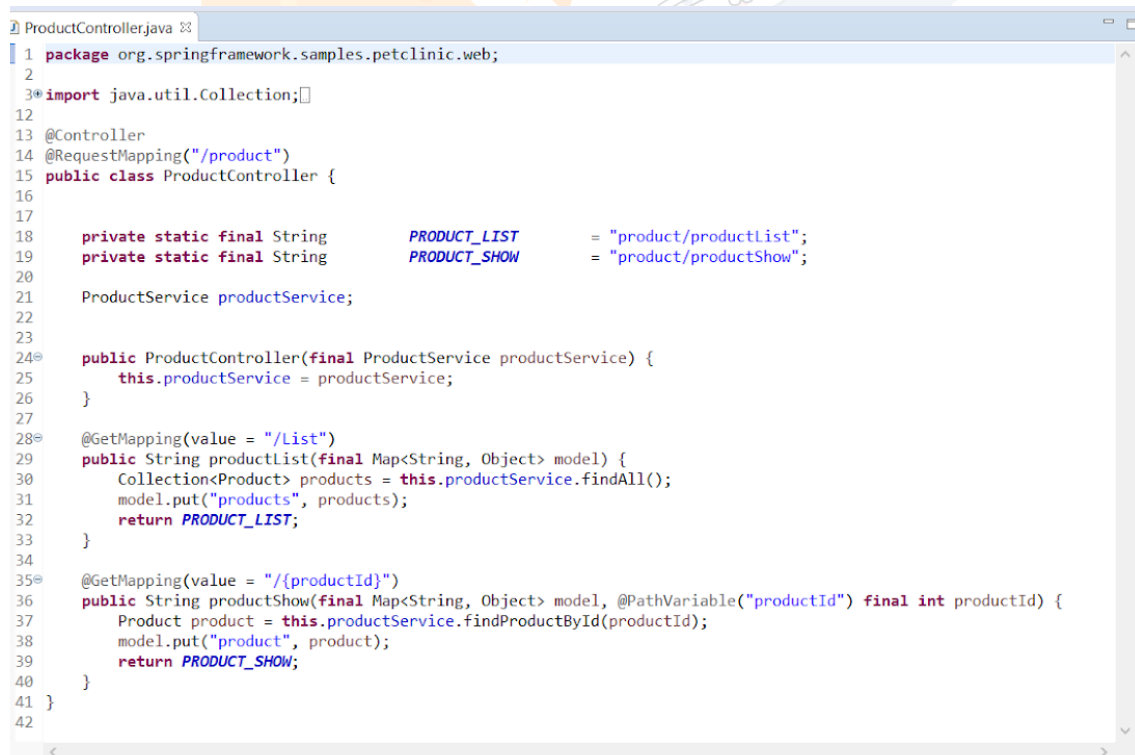
El método findPet es un método situado en el controlador de Pet que no es utilizado y está por tanto obsoleto, es decir, es Dead Code, por lo que para la mejora del código se eliminará para que en el proyecto no existan métodos que no sean de utilidad o están completamente obsoletos, ya que no se les da un uso desde hace un largo tiempo.

```
PetController.java x
68     }
69
70     @ModelAttribute("owner")
71     public Owner findOwner(@PathVariable("ownerId") final int ownerId) {
72         return this.ownerService.findOwnerById(ownerId);
73     }
74
75     @ModelAttribute("pet")
76     public Pet findPet(@PathVariable("petId") final Integer petId) {
77         Pet result = null;
78         if (petId != null) {
79             result = this.petService.findPetById(petId);
80         } else {
81             result = new Pet();
82         }
83         return result;
84     }
85
86     @InitBinder("owner")
87     public void initOwnerBinder(final WebDataBinder dataBinder) {
88         dataBinder.setDisallowedFields("id");
89     }
90
91     @InitBinder("pet")
92     public void initPetBinder(final WebDataBinder dataBinder) {
93         dataBinder.setValidator(new PetValidator());
94     }
```

Product

Tenemos una clase llamada Product. Dicha clase tiene un controlador asociado en el que solo hay un List y un Show orientado a que las personas con rol de Owner, ya que la parte de mostrarle esto mismo al administrador se encuentra en el AdminController. Esta clase, por tanto, la podemos considerar una Lazy class, ya que, para lo poco que se hace en la clase y teniendo en cuenta que parte de lo relacionado con Product se realiza en otros controladores, esta clase no debería estar, y sus métodos deberían estar en el OwnerController. Obviamente, todas las pruebas que estén en clases de ProductTest habrá que eliminarlas y también habrá que incluir su contenido en las clases de prueba de Owner.

Captura de las modificaciones:



```
1 package org.springframework.samples.petclinic.web;
2
3 import java.util.Collection;
4
5 @Controller
6 @RequestMapping("/product")
7 public class ProductController {
8
9     private static final String PRODUCT_LIST = "product/productList";
10    private static final String PRODUCT_SHOW = "product/productShow";
11
12    ProductService productService;
13
14    public ProductController(final ProductService productService) {
15        this.productService = productService;
16    }
17
18    @GetMapping(value = "/list")
19    public String productList(final Map<String, Object> model) {
20        Collection<Product> products = this.productService.findAll();
21        model.put("products", products);
22        return PRODUCT_LIST;
23    }
24
25    @GetMapping(value =("/{productId}")
26    public String productShow(final Map<String, Object> model, @PathVariable("productId") final int productId) {
27        Product product = this.productService.findProductById(productId);
28        model.put("product", product);
29        return PRODUCT_SHOW;
30    }
31 }
```

Clases que también serán eliminadas y cuyo contenido se pasará a los respectivos de Owner:

```
ProductControllerE2ETest.java ProductServiceTests.java ProductControllerTests.java
1
2 package org.springframework.samples.petclinic.web;
3
4 import org.hamcrest.Matchers;
5
21
22 @WebMvcTest(controllers = ProductController.class, excludeFilters = @ComponentScan.Filter(type = FilterType.ASSIGNABLE))
23 public class ProductControllerTests {
24
25     private static final int TEST_PRODUCT_ID = 1;
26
27     @MockBean
28     private ProductService productService;
29
30     @Autowired
31     private MockMvc mockMvc;
32
33     private Product product;
34
35
36     @BeforeEach
37     void setup() {
38         this.product = new Product();
39         this.product.setId(ProductControllerTests.TEST_PRODUCT_ID);
40         this.product.setName("Prueba");
41         this.product.setDescription("Prueba");
42         this.product.setImage("https://www.petpremium.com/wp-content/uploads/2012/10/5-healthy-dog-foods-430x226.jpg");
43         this.product.setPrice(20.0);
44         this.product.setStock(true);
45
46         BDDMockito.given(this.productService.findProductById(ProductControllerTests.TEST_PRODUCT_ID)).willReturn(this.product);
47     }
48
49     @WithMockUser(value = "spring")
50     // ...
51 }
```

Tras la realización de los cambios, hemos eliminado 4 clases en total, únicamente añadiendo dos métodos a otra clase, y sus respectivas pruebas. Ahora el código es más coherente respecto a la clase Product, ya que tiene la parte de Administrador en el AdminController, y la parte de Owner en el OwnerController, permitiéndonos, como he dicho, prescindir de todas las clases de test de Product, y de su controlador. El resultado son clases más largas de tests, pero sin llegar a ser excesiva, además de que con el nombre de los métodos se puede observar perfectamente a que corresponde cada método, lo que no lo hace difícil de entender.



NotificationService

Realizando las pruebas de rendimiento, nos dimos cuenta que el sistema se saturaba con pocas peticiones a la hora de listar las notificaciones para administrador.

Por ello decidimos realizar una factorización para la caché, debido a que los únicos que crean y eliminan son los administradores, por lo que, los listados se almacenarían en caché hasta que se creara una nueva notificación.

Para ello se han creado dos clases nuevas:

CacheConfiguration.java

```
package org.springframework.samples.petclinic.configuration;

import org.springframework.cache.annotation.EnableCaching;

@Configuration
@EnableCaching
public class CacheConfiguration {
}
```

CacheLogger.java

```
package org.springframework.samples.petclinic.configuration;

import org.ehcache.event.CacheEvent;
import org.ehcache.event.CacheEventListener;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class CacheLogger implements CacheEventListener<Object, Object> {
    private final Logger LOG = LoggerFactory.getLogger(CacheLogger.class);

    @Override
    public void onEvent(CacheEvent<?, ?> cacheEvent) {
        LOG.info("Key: {} | EventType: {} | Old value: {} | New value: {}",
            cacheEvent.getKey(), cacheEvent.getType(), cacheEvent.getOldValue(),
            cacheEvent.getNewValue());
    }
}
```

El archivo ehcache3.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<config
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://www.ehcache.org/v3"
    xsi:schemaLocation="
        http://www.ehcache.org/v3
        http://www.ehcache.org/schema/ehcache-core-3.7.xsd">
    <!-- Persistent cache directory -->
    <!--<persistence directory="spring-boot-ehcache/cache" />-->
    <!-- Default cache template -->
    <cache-template name="default">
        <expiry>
            <ttl unit="seconds">120</ttl>
        </expiry>
        <listeners>
            <listener>
                <class>org.springframework.samples.petclinic.configuration.CacheLogger</class>
                <event-firing-mode>ASYNCHRONOUS</event-firing-mode>
                <event-ordering-mode>UNORDERED</event-ordering-mode>
                <events-to-fire-on>CREATED</events-to-fire-on>
                <events-to-fire-on>EXPIRED</events-to-fire-on>
                <events-to-fire-on>EVICTED</events-to-fire-on>
            </listener>
        </listeners>
        <resources>
            <heap>1000</heap>
        </resources>
    </cache-template>

    <cache alias="findAll" uses-template="default">
        <key-type>org.springframework.cache.interceptor.SimpleKey</key-type>
        <value-type>java.util.ArrayList</value-type>
    </cache>
</config>
```

Luego se añadió una nueva línea en application.properties

```
35 # Maximum time static resources should be cached
36 spring.resources.cache.cachecontrol.max-age=12h
37 spring.cache.jcache.config=classpath:ehcache3.xml
38
```

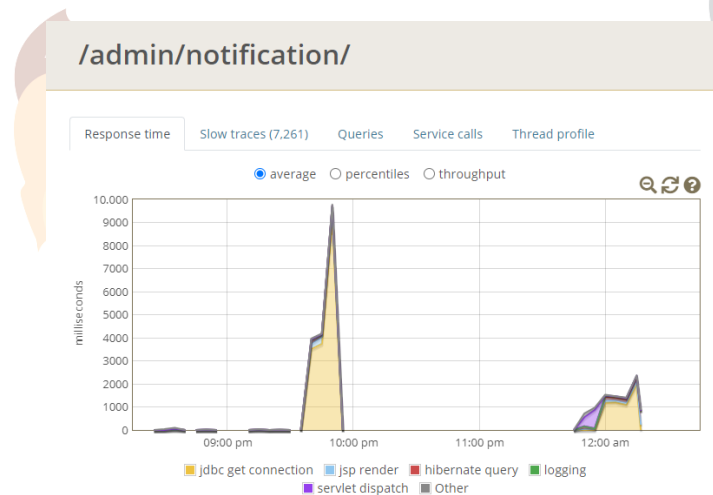
Hemos modificado las anotaciones de dos métodos del NotificationService:

```
@Transactional
@CacheEvict(cacheNames="findAll", allEntries=true)
public void save(final Notification notification) {
    this.repository.save(notification);
}

@Transactional(readOnly = true)
@Cacheable("findAll")
public Iterable<Notification> findAll() throws DataAccessException {
    return this.repository.findAll();
}
```

Con todas estas modificaciones y añadidos, conseguimos que las notificaciones se guarden en caché hasta que se crea una nueva, lo que disminuye notablemente el tráfico de nuestra aplicación.

Para ellos, una vez realizadas las modificaciones, volvimos a lanzar la prueba de rendimiento.



El primer pico es la prueba de rendimiento sin llevar a cabo ninguna modificación, mientras que el segundo pico, es el lanzamiento de nuevo de la prueba de rendimiento de la HU.09, y podemos observar un descenso drástico del tráfico.